PLANT GROWTH SYSTEM
EXTENDED USER MANUAL

Version: 1.0	**Date:** February 17, 2026
Project Title: Plant Growth System	**Document:** Extended User Manual

CONTENT

INTRODUCTION

Purpose of the Tutorial

The purpose of this extended tutorial is to gain a deep understanding of how the system works so you can confidently create your own **Plant Prefabs** from scratch.

By the end of this guide, you will:

- Understand how PlantEntities and PlantStates interact
- Configure actions and conditions inside prefab-based states
- Create your own custom plant behavior
- Use the Demo Scene to observe your plant evolving in real time

This tutorial is not just about following steps — it is about understanding the logic behind the simulation so you can design plants that behave exactly as you intend.

What You Will Learn

In this guide, you will learn how to:

- Configure biological states (PlantStates)
- Adjust variables, thresholds, and transitions
- Define growth logic using conditions and actions
- Test and validate your plant using the Demo Scene

Step 1 – Preparing the Environment

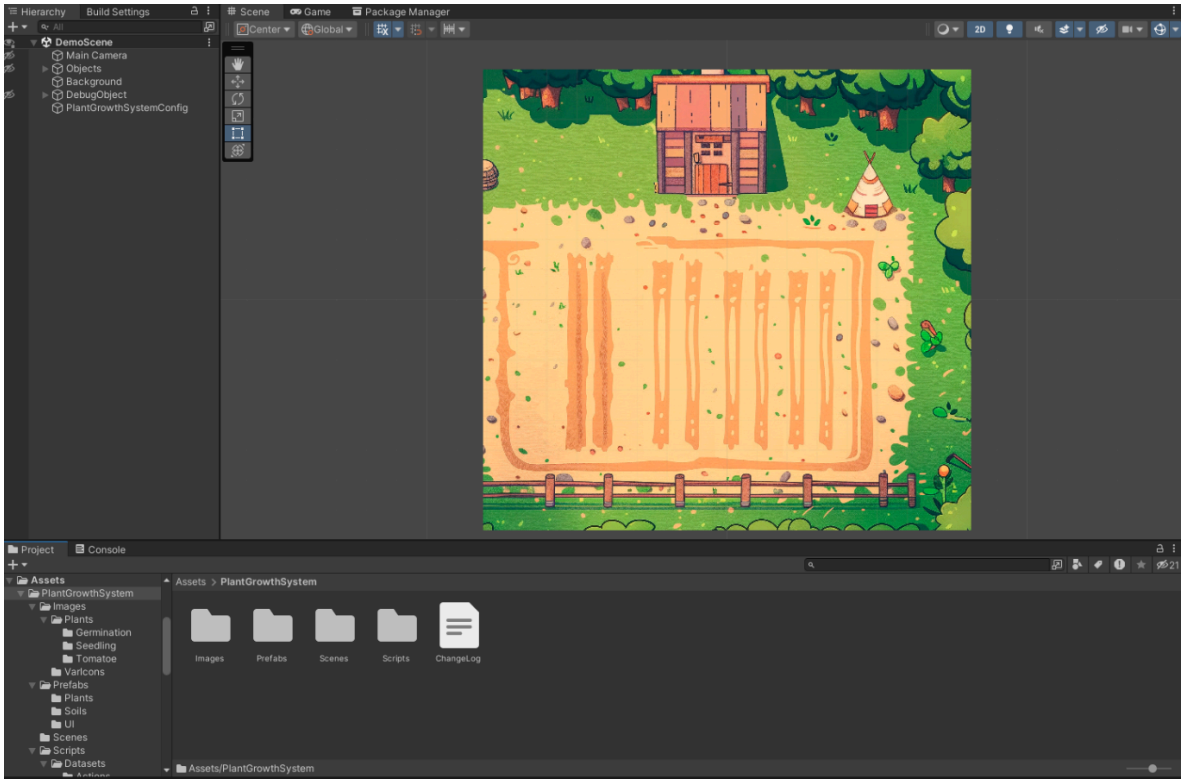
Before creating your own plant, we need to prepare the correct working environment.

1. Open the Demo Scene

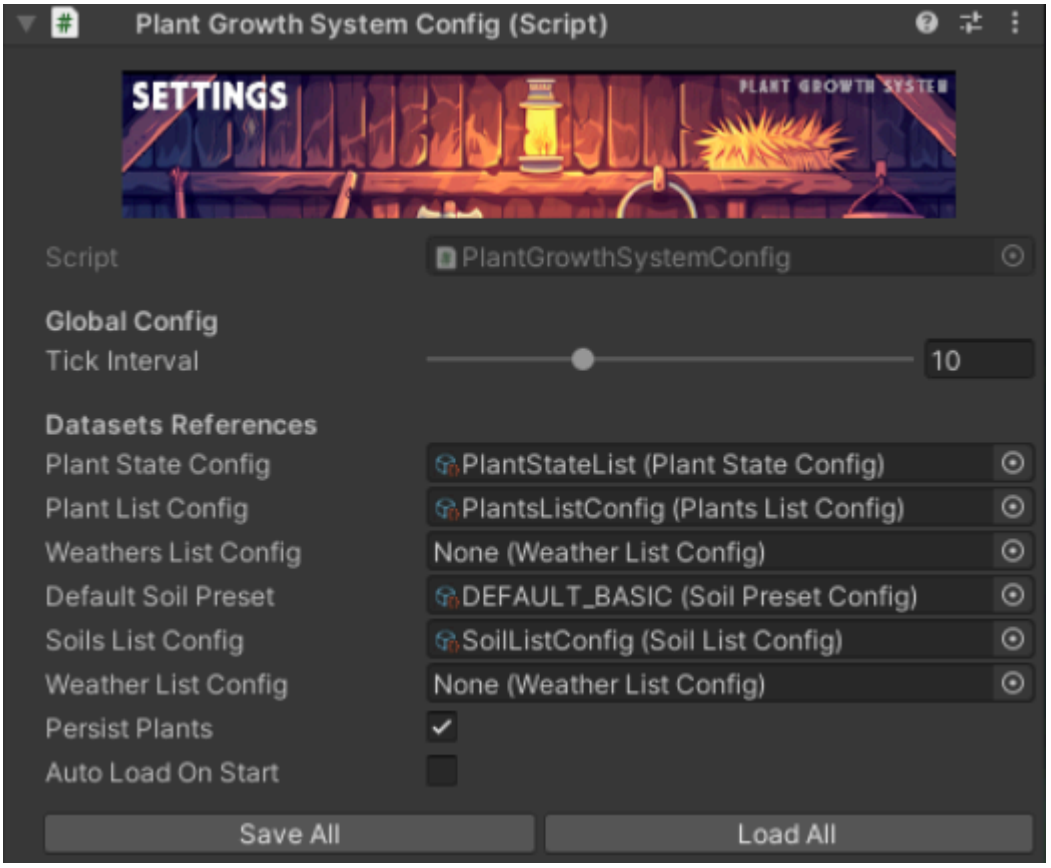
Start by opening the **Demo Scene** included in the package.

In this scene you will find:

- A small landscape used as a testing ground
- The demo interface, which helps visualize variables and simulation changes in real time
- Most importantly: the global configuration object **PlantGrowthSystemConfig**



Demo Scene



The Settings Object

The **PlantGrowthSystemConfig** object contains the global configuration of the simulation..

Below is a brief overview of its main fields:

Tick Interval

Defines the simulation cycle timing.

This value controls how often the system:

- Executes cyclic actions
- Evaluates action condition sets

In short, this determines the speed of the simulation.

Plant State Config

Contains the predefined biological PlantState types available in the system.

These define the possible biological states that plants can use, but the actual behavior is configured inside the Plant Prefabs.

Plant List

A centralized list of Plant Prefabs.

This is mainly used to:

- Keep all plant prefabs organized in one place
- Make them available to the simulator and Demo Scene

Only prefabs added here can be instantiated through the Demo interface.

Weather List Config

A dataset containing available Weather Prefabs.

Works the same way as the Plant List, but for atmospheric simulation objects.

Soils List Config

A dataset containing available Soil Prefabs.

Only Soils registered here will be available for instantiation in the Demo Scene.

Persist Plants

Boolean value.

If enabled, plant variable values are saved and can be restored later.
Useful for long-running simulations or testing persistence behavior.

Auto Load on Start

Boolean value.

If enabled, previously saved plants are automatically loaded when entering Play Mode in Unity.

STEP 2 Creating a Soil

Before creating a plant, we need a **Soil**.

What Is a Soil?

A **Soil** represents a simulation “plot” — a defined piece of land where plants can be placed and interact with environmental conditions.

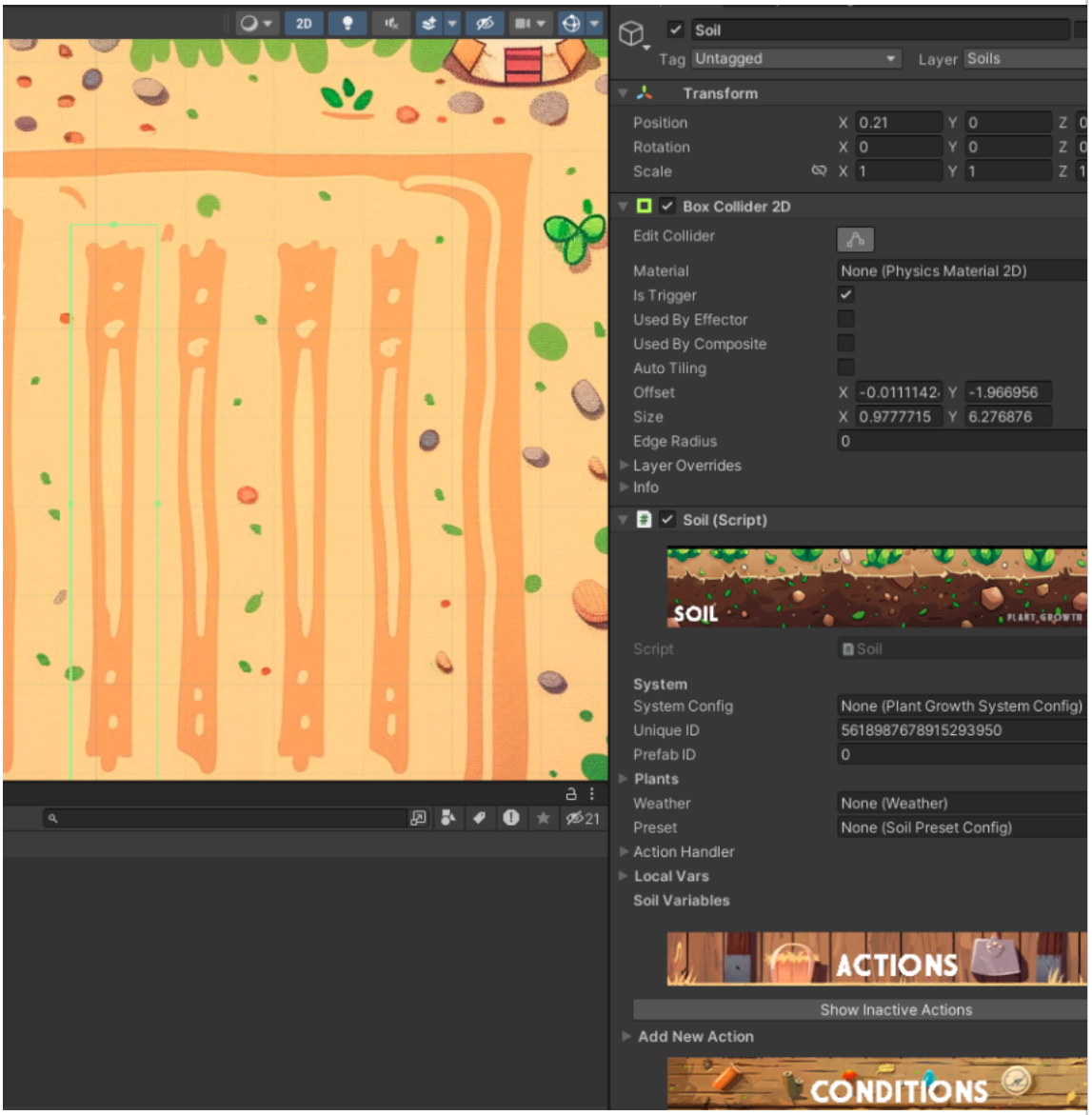
It is not just visual ground.
It is a simulation entity responsible for storing and modifying environmental variables such as moisture and soil-related data.

In order to allow interaction in the Demo Scene, a Soil requires a **Collider2D**, which enables plant placement through user clicks.

Creating the Soil GameObject

To create a Soil:

1. Create a new **GameObject** in your scene.
2. Add the **Soil** component.
3. Assign it to a dedicated **Soils Layer**.
4. Add an appropriate **Collider2D** (BoxCollider2D, PolygonCollider2D, etc.).
5. Adjust the collider to match the desired planting area.



It is recommended to configure separate Layers for:

- Soils
- Plants

This helps with raycasting, interaction handling, and logical separation inside your project.

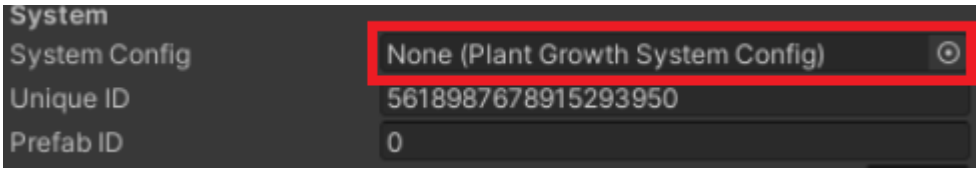
Important Reminder

After creating the Soil, make sure it has a reference to the **PlantGrowthSystemConfig** object.

In most cases, the system will detect and assign it automatically.
However, if it is not assigned:

- Locate the **Config reference field** inside the Soil component
- Drag and drop the **PlantGrowthSystemConfig** object from the scene

Without this reference, the Soil will not properly participate in the simulation tick or access the required datasets.



Architectural Concept: Values vs Execution

The system can be understood as having two differentiated parts:

1. Value Storage

Responsible for storing and maintaining variable data.

2. Value Execution

Responsible for modifying those values through actions and conditions during the simulation tick.

Weather and Soil

Both **Weather** and **Soil** contain:

- A value storage layer (their variables)
- An execution layer (actions and condition evaluation via ActionHandler)

They actively participate in the simulation loop.

PlantEntity

The **PlantEntity**, on the other hand, only stores values and shared references.

It does not directly execute actions.

The actual execution logic for plants happens inside their **PlantStates**.

Step 3 – Adding Variables to the Soil

Now that the Soil is created and linked to the **PlantGrowthSystemConfig**, we can add variables to start observing the simulation.

---

1. Select the Soil

- Click on your **Soil GameObject** in the scene or hierarchy.

---

2. Open Local Variables

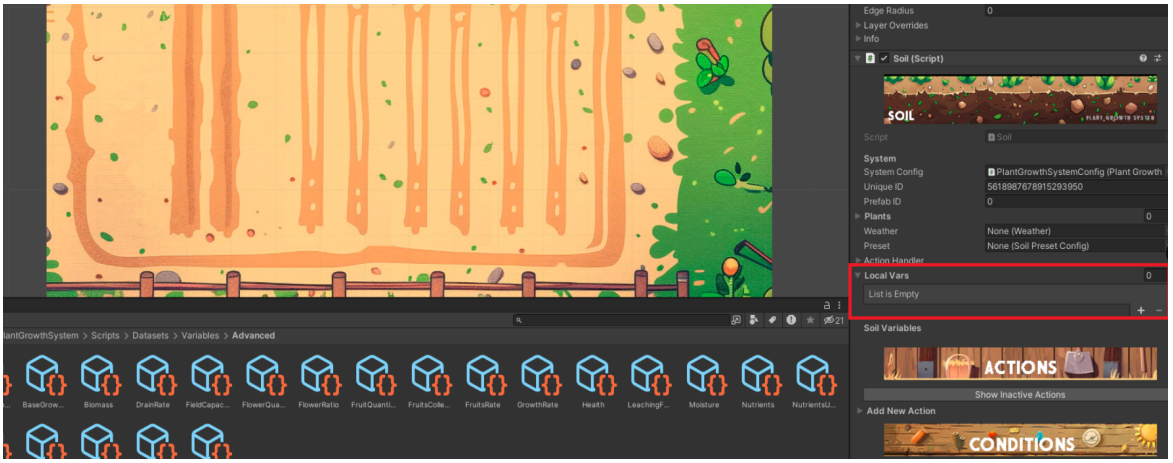
- In the **Soil component**, locate the **Local Vars** section.
- This section stores all variable states specific to this Soil.

3. Add Variables

For this tutorial, we will start simple:

- **Moisture** → Represents the water content of the soil

- **DrainRate** → Controls how quickly moisture decreases over time

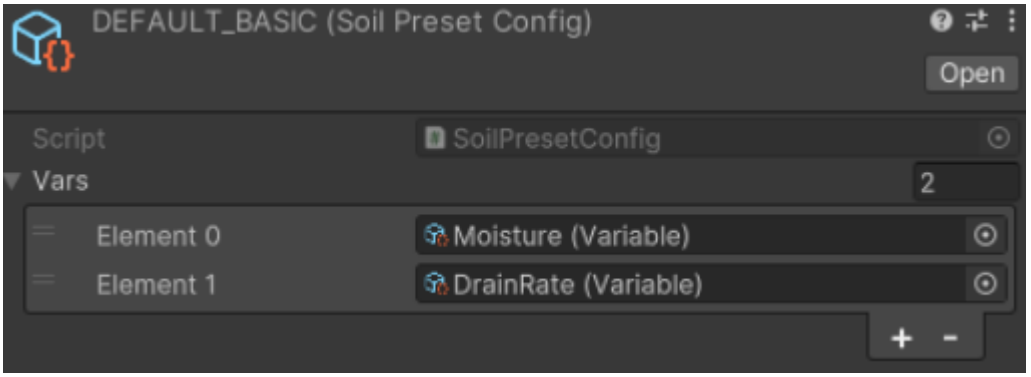


To add these variables:

1. Go to the **Scripts/Datasets/Variables** folder in your project
2. Locate **Moisture** and **DrainRate**
3. Select your Soil, and in the Local Vars array, click the “+” button to add a new element
4. Drag Moisture into the first slot, DrainRate into the second slot

Soil Presets

A Soil Preset, as the name suggests, is a pre-configured Dataset that already contains variables and initial settings.



It is designed to **speed up the creation of new soils** by providing a ready-to-use template.

- Instead of manually adding Moisture, DrainRate, and other variables, you can simply use a preset
- Presets can be duplicated and adjusted to create variations quickly

Note: In future versions, the plan is to expand this system to allow easier plant creation and sharing.
Users will be able to **download and share Soil and Plant Prefabs** via a central database or web portal.

Step 5 – Observing VariableStates at Runtime

Now that you have created a Soil and optionally used a **Soil Preset**, it's time to see how the simulation works in real time.

---

1. Enter Play Mode

- Click **Play** in Unity to start the simulation.

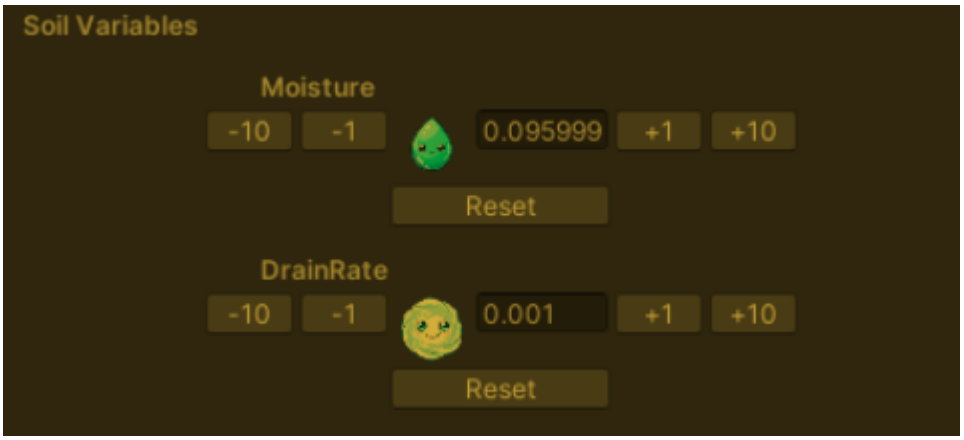
---

2. Inspect the Soil

- Select the Soil GameObject in the scene

At runtime:

- Each variable in **Local Vars** now has a corresponding **VariableState**
- **VariableStates** are instances of the variables for this particular Soil
- They hold the **current values**, which may differ from the defaults defined in the Variable asset



---

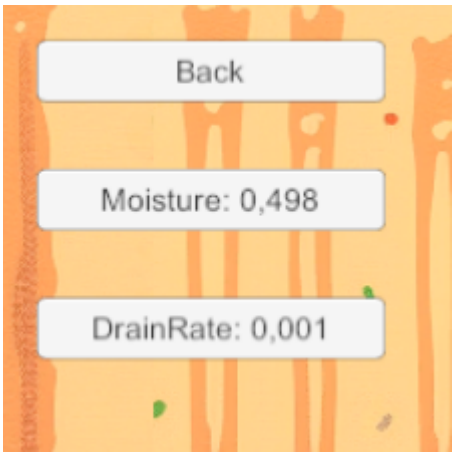
3. Observe Default Values

- The **default values** configured in the Variable asset are automatically applied when the simulation starts
- For example:
    - **Moisture** will be initialized to the default moisture level
    - **DrainRate** will start affecting Moisture according to its settings

4. Interactive Soil Controls

While in **Play Mode**, the Soil is fully interactive through its **Collider**:

1. **Click on the Soil** using the mouse
    - The **Demo interface** will appear, showing all variables and their current **VariableStates**
2. **List Variables**
    - You can see all the variables associated with this Soil
    - This allows you to track Moisture, DrainRate, or any other variables in real time



3. **Modify Values at Runtime**
    - Clicking on a variable allows you to **change its value immediately**
    - Changes will be reflected in the simulation tick and can influence plant growth
4. **Add Plants**
    - Once a plant prefab is ready, clicking on a Soil Collider will give you the option to **add the plant to this Soil**
    - The new plant will automatically initialize its **ActionsOnInit** and begin interacting with the Soil's VariableStates

Step 6 – Adding the First Action

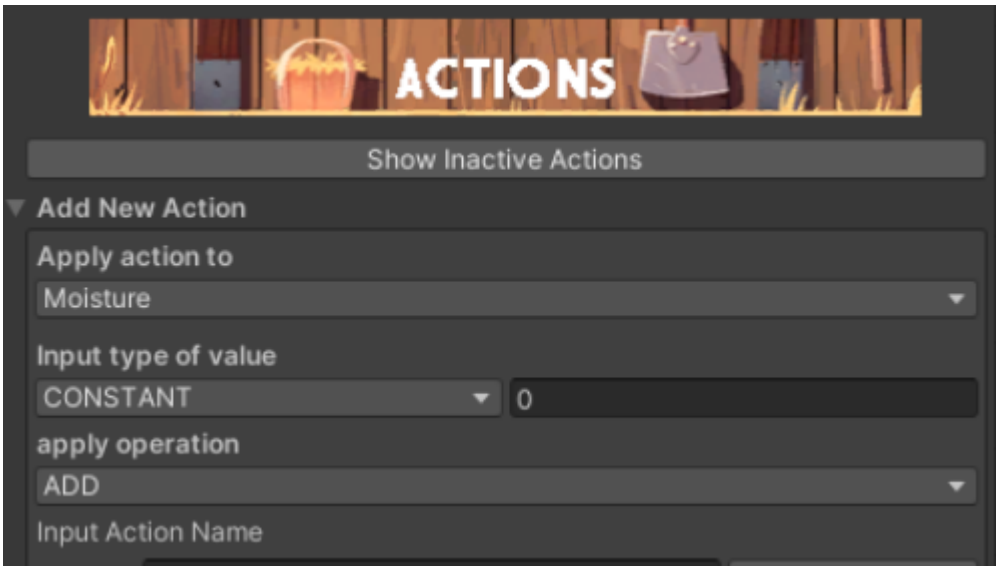
Now that we have a Soil with VariableStates in runtime, it's time to **add an action** to see the system working dynamically.

---

1. Action Menu

- Click on your **Soil GameObject** in the scene hierarchy

- Inside the Soil component, find the **Action** section
- This is where actions can be added to modify variables over time or when conditions are met



---

2. Create a New Action

For this tutorial, we'll add a **simple Moisture Drain action**:

- **Action Operation:** Subtract Value
- **Target Variable:** Moisture
- **Value Source:** DrainRate (Select first a reference type of value)

This means that every tick, the Soil will **reduce its Moisture by the DrainRate** automatically

Click on Add Action.

Active vs Inactive Actions

*In the system, each **action** can be either **Active** or **Inactive**, which defines **when it is executed**.*

Active Actions

- *Executed **every tick** of the simulation automatically*
- *Used for continuous processes, for example:*

    - *Subtracting **DrainRate** from Moisture*
    - *Updating temperature or light effects*
- *The **ActionHandler** will run them automatically during the simulation cycle*

Inactive Actions

- *Not executed every tick*
- *Designed for **one-time or conditional execution**, such as:*
    - ***ActionsOnInit** when a plant or soil is instantiated*
    - *Actions triggered by **Action Condition Sets***
- *Useful for **events that only happen under specific conditions**, without impacting the tick performance*

Managing Actions in the Inspector

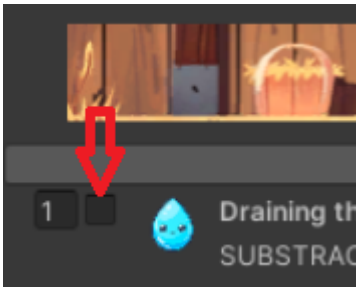
*In the Demo interface or the Soil/Plant inspector, you can **filter or toggle Active and Inactive Actions***
This allows you to focus on the actions currently running or planned for conditional execution

5. Test the Action

1. Active your new Action

Toggling Action Status

- *To make an action Active or Inactive, simply click the checkbox next to its name in the ActionHandler list*
- *The checkbox is located next to the action's ID number*
- *This allows you to instantly switch whether the action runs every tick or only under specific conditions*



Tip: This is a quick way to test different behaviors or temporarily disable actions without deleting them.

2. Enter on Play Mode and observe the Soil's **Moisture VariableState**
3. You should see the value **decrease according to DrainRate** at each tick

Tip: You can also modify **DrainRate in real time** through the Demo interface to see immediate changes in Moisture

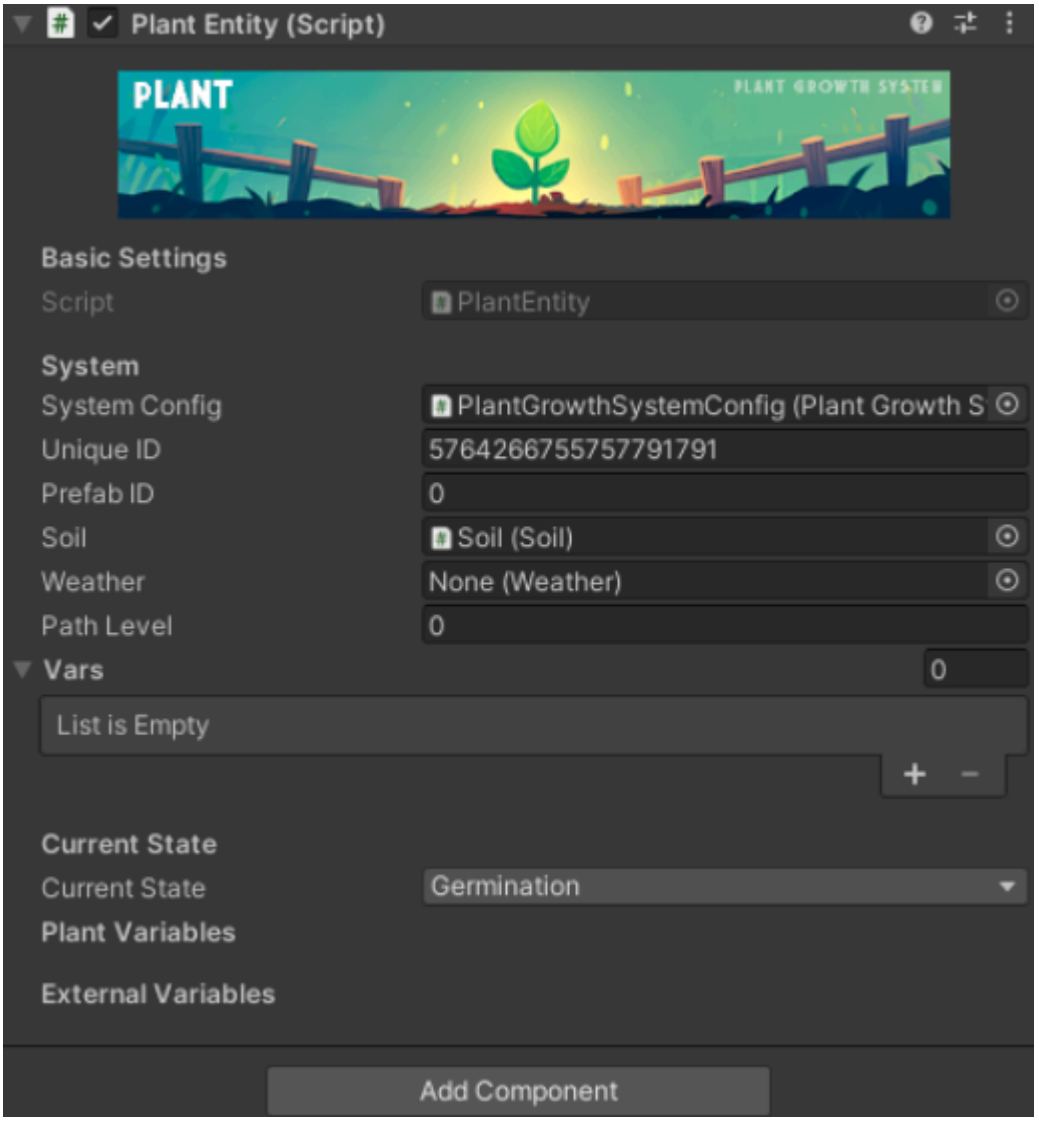
Understanding Plants vs Soil

Before we start adding conditions and ActionConditionSets, it's important to understand the structural difference between Plants and Soils.

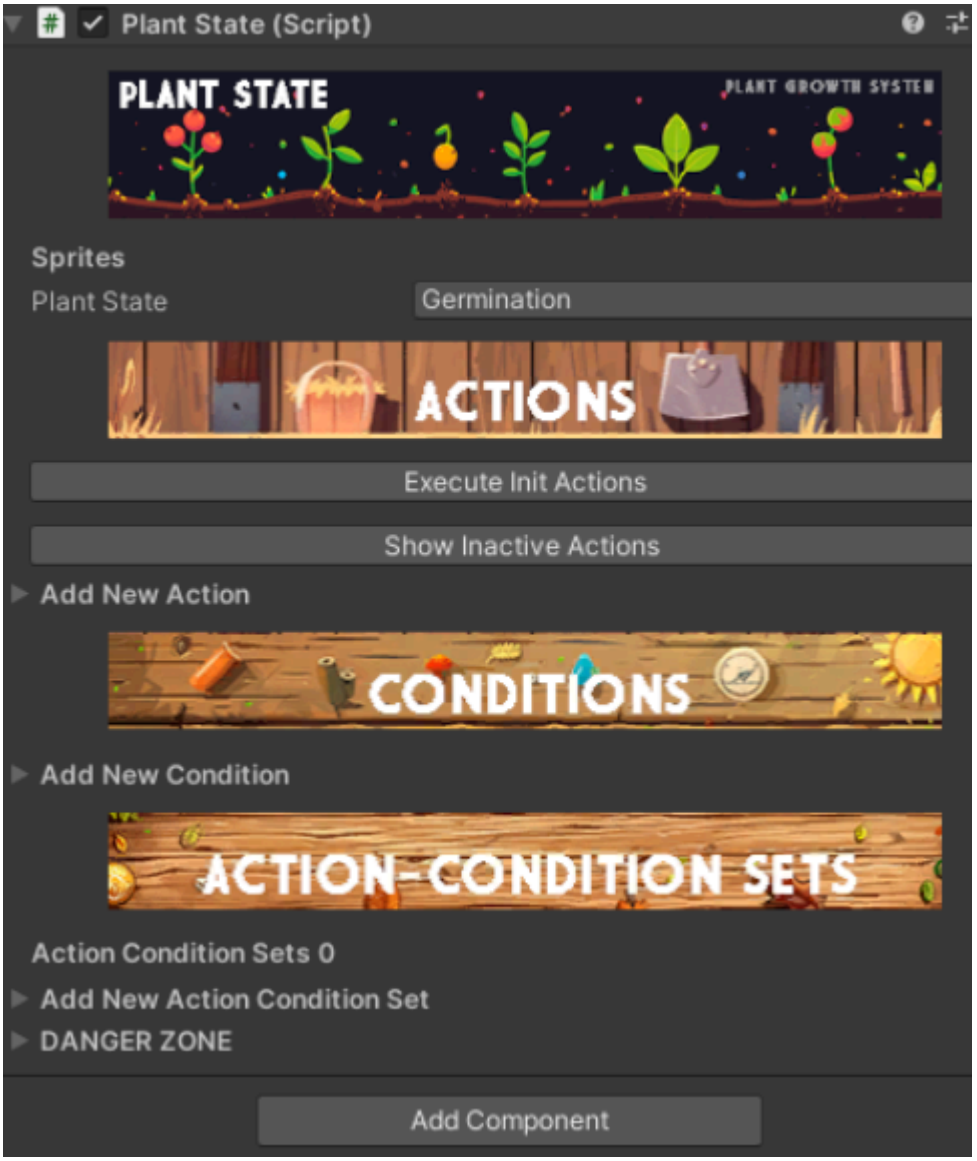
Plant Structure

A Plant in the system is divided into two parts:

1. **PlantEntity**
   The main container that holds the global configuration, references, and datasets
   Does not execute actions directly. Acts as the anchor for all its child states



2. **PlantState (child of PlantEntity)**
   Represents a specific biological growth stage of the plant
   Each PlantState can be selected using a state selector
   Executes its own Actions independently
   Can include sprites, effects, sounds, and other GameObjects
   Becomes active or inactive depending on the growth stage, as part of the StateMachine



Soil vs Plant

- Soil stores variables and executes actions.
- Plants interact with the Soil's VariableStates, but **their actions are handled inside their PlantStates**

- This separation allows:
  - Clean management of growth logic
  - Reusable prefabs
  - Dynamic interaction between environment and plant

Step 7 – Creating the Plant

1. Create the Plant GameObject

- In the Unity hierarchy, **create a new empty GameObject**
- Rename it to something meaningful, like `TomatoPlant` or `Plant_Example`

---

2. Add the PlantEntity Script

- With the GameObject selected, **add the `PlantEntity` script**
- This will make it the **main container** for all growth states and references

---

3. Add PlantStates as Children

- **Create child GameObjects** under the PlantEntity for each growth stage
    - Example: `Germination`, `Flowering`, `Fruiting`
- Add the **PlantState component** to each child
- In the **PlantState selector**, choose the type of state for this stage

Example: Implementing Growth Logic in PlantState

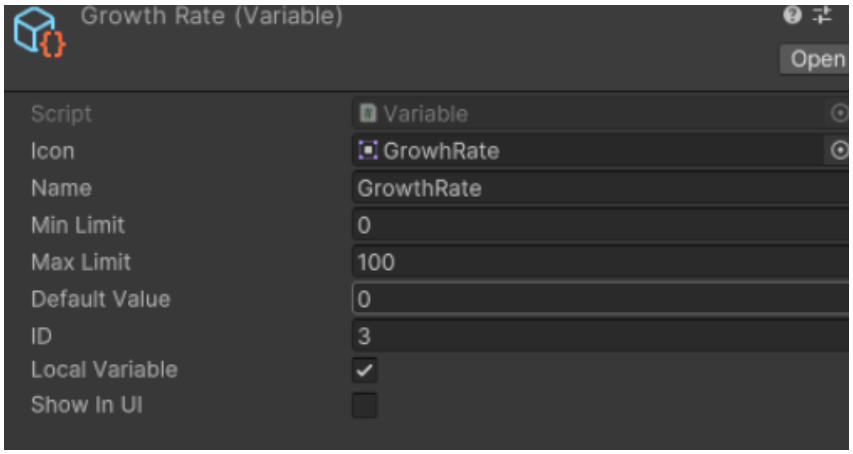
This example shows **how to make your plant grow dynamically**, using **Local Vars, Conditions, and ActionConditionSets**.

---

1. Prepare the First PlantState

- Select your **first PlantState** (e.g., `Germination`)
- Add the following variables to **Local Vars**:
    - `BIOMASS` (accumulative variable)
    - `GROWTHRATE` (Local variable, resets each tick)
    - `BASEGROWTH` (accumulative or constant value to use in calculations)

Note: `GROWTHRATE` is marked as a **Local Var**.
Local Vars are **reset every tick**, useful for temporary calculations before adding to accumulative variables like BIOMASS.



---

2. Create a Condition

We want to grow the plant **only if there is enough soil moisture**:

1. Go to the **Conditions section** in the PlantState inspector

2. Create a new condition:
    - Check that **Soil Moisture > 0.2**
    - This ensures that growth only occurs when the soil has at least 20% moisture

---

3. Set Up the ActionConditionSet

- Create a new **ActionConditionSet** and assign the condition created above
- Add an **action** within this set:
    - **Action Type:** Add Value
    - **Target Variable:** GROWTHRATE
    - **Value Source:** BASEGROWTH

  Because GROWTHRATE is a **Local Var**, it will **reset to 0 at the beginning of the next tick**, ensuring growth only occurs while the condition is true.

---

4. Apply Continuous Growth

- Add a **cyclic Active Action** to the PlantState:
    - **Action Type:** Add Value
    - **Target Variable:** BIOMASS
    - **Value Source:** GROWTHRATE

This will **accumulate the growth** each tick, using the temporary value calculated in the previous step.

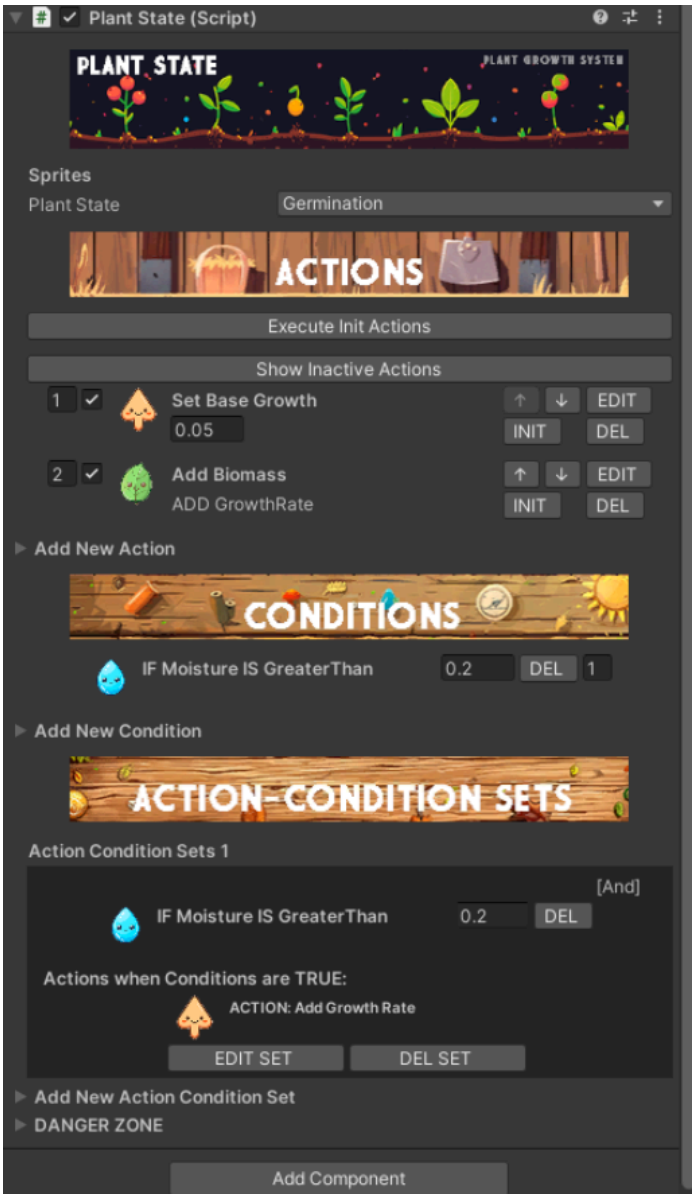
- When the moisture condition is met, GROWTHRATE is positive → BIOMASS increases
- When the condition fails, GROWTHRATE resets → no growth occurs

---

5. Visualizing in the Inspector

- The **Inspector is highly visual**:
    - You can see all **Variables, Actions, and Conditions** linked together
    - The **ActionConditionSets** show which actions are executed when conditions are met
    - Active and Inactive actions are clearly labeled and toggleable with checkboxes

Tip: This setup allows you to **add multiple PlantStates** with different growth logics, sprites, effects, or sounds, creating a fully dynamic and realistic plant lifecycle.



Setting Constant Values

*You can **set a constant value** (e.g., `BASEGROWTH = 0.05`) in two ways:*

1. ***Action without conditions** → always executes every tick*
2. ***ActionOnInit** → executes once when the plant is instantiated (Click Init Button on the inspector)*

*Tip: Use **OnInit** for initialization, or an unconditional action if you want the value to be applied continuously. I've used Set Base Growth Action to Set Growth to 0,05.*

Community & Shared Content

This plugin is designed to grow alongside its community.

The long-term goal is to provide:

- A shared Plant Prefab database
- A collection of reusable Soils, Plants, and configurations
- Community-driven improvements and discussions

All prefabs will be fully compatible with the simulation system and can be downloaded and used directly in your projects.

Discord Server

https://discord.gg/DPWHUP7vWA

A dedicated Discord server is available for:

- Questions and troubleshooting
- Suggestions and feature requests
- Sharing Plant and Soil Prefabs
- Discussing simulation setups and design ideas

Community feedback will directly influence future updates of the plugin.